ISSN 0265-2919
80p 13

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO

AND MICRO SHALL SPEAK UNTO MICRO

An ORBIS Publication
IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

Next Week

The attractive Atmos is the successor to the Oric 1. It boasts a quality keyboard, 48K of memory, colour graphics and has its own disk drive and printer.

Word processing is a popular application for business machines but it can prove tricky on home micros. We look at this important feature on the popular micros.

COMING SOON – A course on machine code programming for the 6809 microprocessor as used in the Dragon and Tandy Color computers.

QUIZ

1) What is wrong with the line: 225 CLS in BASICODE?

2) Why do records in a random access file have to be of the same length?

3) What is the name of the process by which a computer controls the individual bits that make up the machine's screen memory?

4) Why did Camputers not have to worry too much about software for the Lynx Laureate?

Answers To Last Week's Quiz

A1) By removing the barrel, more ambient light is allowed to enter the photo-diode, therefore making it more difficult to focus on the screen.

A2) Feedback is useful to a computer when it needs to 'sense' the result of its actions on a peripheral device.

A3) A NOR gate will return a result of one if the input is zero and vice versa, it is therefore ideal as a switching device.

A4) The binary number to switch bit 7 to zero is 01111111. This would require an AND operation.

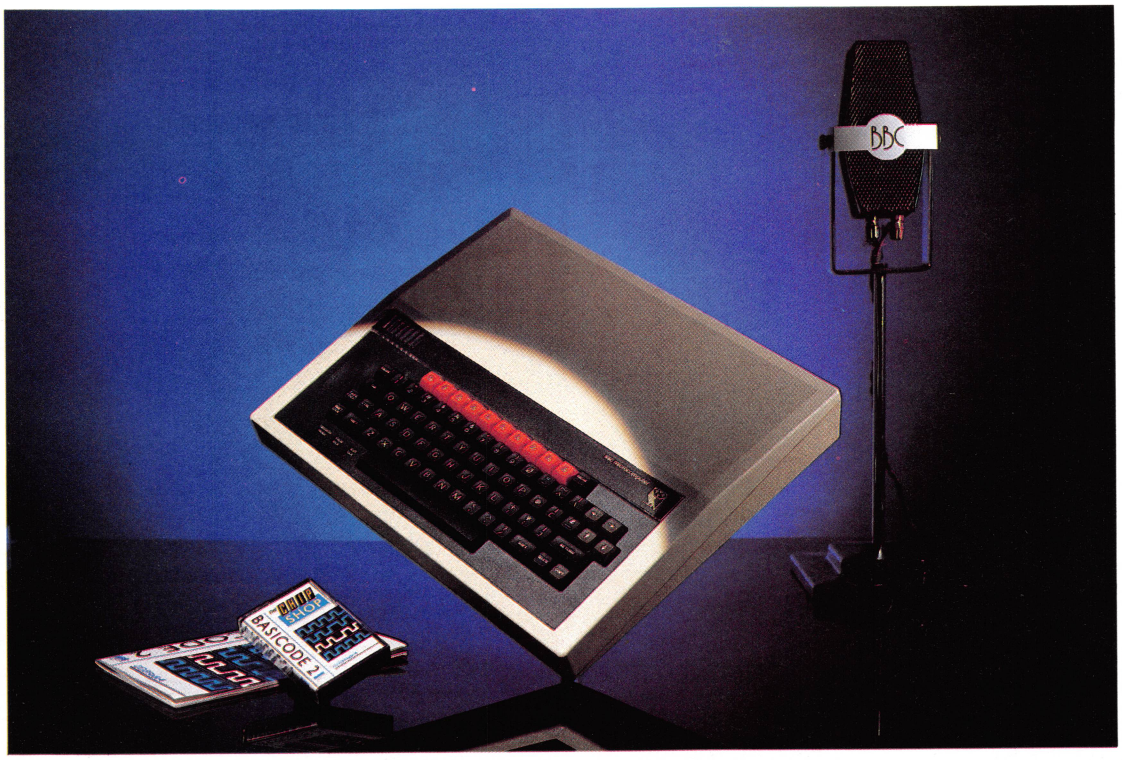
QUIZ

COVER PHOTOGRAPHY BY IAN McKINNELL

Editor Jim Lennox; **Art Director** David Whelan; **Technical Editor** Brian Morris; **Production Editor** Catherine Cardwell; **Picture Editor** Claudia Zeff; **Sub Editor** Robert Pickering; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Editorial Assistant** Stephen Malone; **Contributors** Steven Colwill, Max Phillips, Gareth Jefferson, Richard Pawson; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Co-ordinator** Ian Paton; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 85 Charlotte Street, London W1P 1LB; **© APSIF Copenhagen 1984; © Orbis Publishing Ltd 1984: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

HOME COMPUTER ADVANCED COURSE – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95
How to obtain your copies of HOME COMPUTER ADVANCED COURSE – Copies are obtainable by placing a regular order at your newsagent, or by taking out a subscription. Subscription rates: for six months (26 issues) £23.80; for one year (52 issues) £47.60. Send your order and remittance to Punch Subscription Services, Watling Street, Bletchley, Milton Keynes, Bucks MK2 2BW, being sure to state the number of the first issue required.
Back Numbers UK and Eire – Back numbers are obtainable from your newsagent or from HOME COMPUTER ADVANCED COURSE. Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER ADVANCED COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.
How to obtain binders for HOME COMPUTER ADVANCED COURSE – UK and Eire: Please send £3.95 per binder if you do not wish to take advantage of our special offer detailed in Issues 5, 6 and 7. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER ADVANCED COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER ADVANCED COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.
Note – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

COMMON DENOMINATOR



IAN McKINNELL

Common Standard
BASICODE enables micros to communicate with each other through a common standard. It uses a minimum set of BASIC commands and its own tape format to allow a dozen or so micros to swap programs. BASICODE programs are even transmitted by radio stations, allowing listeners with different micros to use the same programs

The main obstacle facing most home computer users who wish to exchange software is that of machine compatibility. Software written for one make of computer will not, as a rule, run on another. However, BASICODE, a language recently developed in the Netherlands, is a new approach to solving this perennial problem.

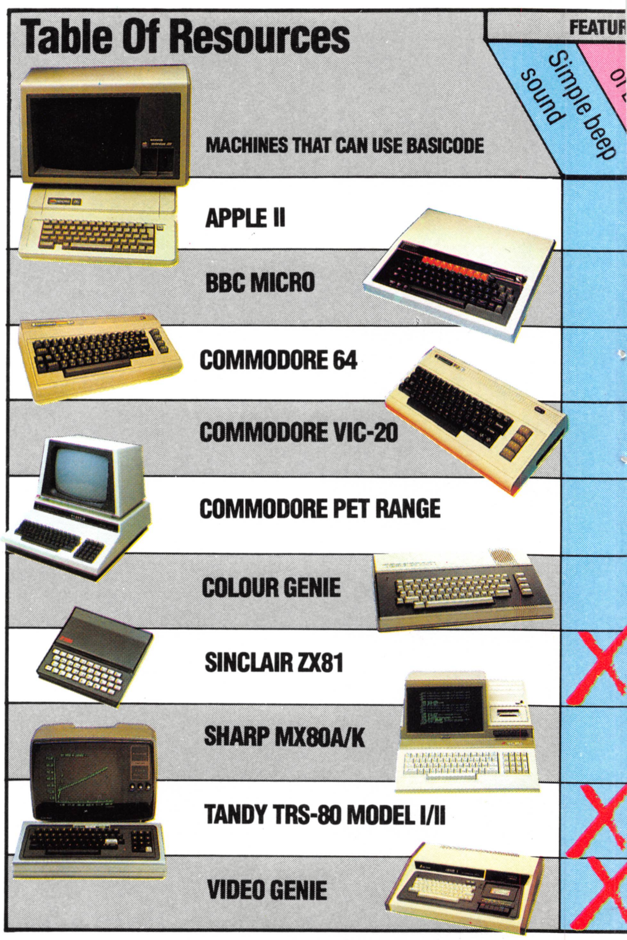
BASIC has now firmly established itself as the standard language for home micros. However, as every home computer user knows, there are enormous variations in the dialects available. Even when machines do share a common dialect, such as Microsoft BASIC, there is no guarantee that a program written on one type of computer will necessarily run on a different model.

There are now signs that this may be about to change. At the beginning of 1984, BBC Radio's *Chip Shop* programme began transmitting programs in a single dialect of BASIC known as BASICODE, and these programs have been successfully loaded and run by a wide variety of different computers.

BASICODE is a new approach to the problem of compatibility. It was first developed in the Netherlands for use on *Hobbyscoop*, a science and technology programme produced by Teleac, the Dutch equivalent of the Open University. When *Hobbyscoop* first began broadcasting programs in 1978, the programme based its transmissions on the four most popular machines then available — the Apple, Exidy Sorcerer, Commodore PET and Tandy TRS-80. There could be only one transmission for a single machine each week and, as two of these computers had extremely low data transmission rates, listeners had to endure up to eight minutes of screeching. Clearly, this state of affairs was unsatisfactory, and as new machines came onto the market, each requiring its own broadcast, this method of programming transmission was obviously becoming impossible.

The problem was first tackled by an amateur radio enthusiast called Klaas Robers, who produced the first version of BASICODE. This was based on a common subset of BASIC commands, which could be understood by all types of computer. There were teething troubles with the new system. Although different types of machine had identical commands, the computers had different methods of executing them and so the standardisation broke down. So, together with Jochem Herrman, Klaas Robers developed an improved version of the language, known as BASICODE-2.

The first broadcasts of BASICODE-2 were made on New Year's Day 1983, and soon proved to be a



This diagram shows how each of the machines catered for in BASICODE conforms to the format. Those machines with specifications that do not measure up to the standard are marked with a cross. Machines marked with a tick, have facilities beyond what is allowed by BASICODE

success. Listeners as far away as Belgium, France, England, Germany and Denmark reported success in loading the programs. This international attention was increased when the Dutch broadcasting service began transmitting BASICODE-2 on its external network.

BASICODE is founded on the 42 keywords and 11 symbols that most of the machines able to make use of the language have in common. A BASIC keyword is not stored as its component characters but is held in the form of a single-byte token, a symbol which represents the keyword. For example, the keyword LEFT$ is represented on the Commodore 64 by a single byte containing the value 200, rather than by five bytes containing the ASCII values for L, E, F, T and $. This makes the work of the BASIC interpreter much more efficient and uses far less RAM. However, although every computer uses tokenisation for storing and interpreting a BASIC program, each machine uses different values for its tokens. The problem was solved by providing two translation programs, BASICODE-Save and BASICODE-Load. After a program has been written in BASIC, it is SAVEd by using the BASICODE-Save program, which substitutes BASICODE standard tokens for the computer's own BASIC tokens and produces on tape a standard BASICODE program. This program can then be loaded onto another machine using the BASICODE-Load program, which substitutes the machine's BASIC tokens for the BASICODE version.

This raises a major question — how to ensure that the various types of computer read and write to tape in the same way. Once again, although all the machines use the same principle for LOADing and SAVEing programs on tape, in practice a program tape produced by one computer might be very different from another machine's tape. Not only may the data be written and read to the tape at different transmission rates, but also the start and stop bits (the markers that tell the computer where the data begins and ends), and the checksum methods (the system whereby the machine checks that data has been transmitted correctly), may also differ radically. The solution adopted was to suppress the individual machine's own tape handling methods and impose a common *audiocode* format for transmission.

In this format, data is transmitted at 1,200 bits per second. Each byte of data, preceded by a start bit (value 0), is transmitted least significant bit first, and followed by two stop bits (both with a value of 1). For example, the ASCII value of 'A' is 65 — 01000001 in binary — and this would be transmitted in audiocode as 01000001011. A leader marker, consisting of a sequence of stop bits transmitted for five seconds, indicates the start of a BASICODE program. This is then followed by the code for 'start text' (82 in hexadecimal). The BASICODE program is followed by a checksum byte, which enables the computer to check the accuracy of the transmitted data. Another five-second stop bit sequence indicates the end of data transmission.

Although nearly all machines can be adapted to BASICODE by software alone, the TRS-80 models I and III and the Video Genie require a small interface to be added to them in order to allow tapes to be read in correctly. The handbook provided with the Basicode-2 cassette gives full details on how to construct the interface. For the less DIY-inclined, a printed circuit board can be obtained from the Netherlands TRS-80 users group.

In order to write a BASICODE program, you must first load the BASICODE-Save program. This program not only allows the newly-written code to be SAVEd on cassette in a standard format, but also provides a list of subroutines for the computer that are unique to that particular machine. These routines are stored between lines 0 and 999 and, therefore, are unavailable to the programmer.

The reason why these routines are provided by the BASICODE-2 translation program is because a command common to several machines — such as the instruction to clear the screen (CLS) — may be executed in different ways. Instead of using the CLS command, the programmer uses GOSUB 100, which refers to the BASICODE subroutine that performs this function.

The first line a programmer writes should be in the form:

```
1000 A=(value): GOTO 20: REM program name
```

where (value) is the maximum number of characters used by all the strings together. From this point onwards, the user is given a free hand to program as desired. There are, however, a number of

Operation Commands
This is a list of commands and operations that are allowed by BASICODE. Note that many machines will have a far greater number of keywords within the BASIC interpreter that are not recognised within BASICODE

ABS	NEXT
AND	NOT
ASC	ON
ATN	OR
CHR$	PRINT
COS	READ
DATA	REM
DIM	RESTORE
END	RETURN
EXP	RIGHT$
FOR	RUN
GOSUB	SGN
GOTO	SIN
IF	SQR
INPUT	STEP
INT	STOP
LEFT$	TAB
LEN	TAN
LET	THEN
LOG	TO
MID$	VAL
+	<
–	>
*	<>
/	<=
∧	>=

OF BASICODE			COMMANDS NOT IN BASICODE			
imple core IC	Standard tape loading	40×24 text	High res graphics	Colour	Full sound	Structured programming
			✓	✓		
			✓	✓	✓	✓
				✓		
		✗		✓		
			✓	✓	✓	
		✗				
					✓	
	✗	✗				
	✗	✗				

restrictions imposed on the format of the code. For example, variables must be initialised before any operations can be performed on them, so before the command LET T=T+1 can be executed, T must first be set to zero.

There are also some limitations on the use of several of the BASIC keywords. For example:

```
5000 INPUT "PASSWORD?";A$
```

is forbidden by BASICODE-2. The correct format of this line is:

```
5000 PRINT "PASSWORD?": INPUT A$
```

In addition, a program line may not exceed 60 characters in length, and the screen size is assumed to be 24 lines of 40 characters.

It is worth considering at this point why all these restrictions are necessary. In order to include as many machines as possible within the scope of BASICODE there was a 'lowest common denominator' approach to the design. Inevitably there was a trade-off between the sophistication of BASICODE and the number of computers able to make use of it. This led to a certain amount of 'levelling down' to the capabilities of the weaker machines, thus resulting in the limitations placed on more advanced models.

For example, a number of features that the home micro user may look for when purchasing a computer are not catered for in the BASICODE format. There is no facility built into the system to vary the pitch and duration of sounds. There is only the rather primitive BEEP command to work with. Similarly, BASICODE only allows the programmer to produce graphics in low resolution mode. Even these can only be programmed in black-and-white.

Another problem is that since BASICODE was first invented there has been a great deal of progress in the development of BASIC's structured programming techniques. There is no allowance made by BASICODE for case statements such as WHILE...WEND or even a DEF FN command. Structuring is left to the GOSUB command on which the protocol of the language largely depends.

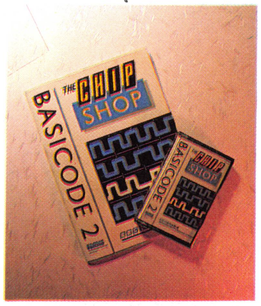
On the other hand, it is worth noting that despite these restrictions, some of the machines supported by BASICODE are unable to live up to even the modest standards set by the protocol. For example, the Vic-20, ZX81, TRS-80 and Video Genie all support displays that are less than the 40×24 character standard.

However, the dedicated programmer should find programming in BASICODE to be a challenge. Because the rules are restrictive, a great deal of care has to be taken by the programmer to ensure that the program written is portable. The programmer has to remember to stay strictly within the 50 or so keywords and operators that the language uses, and use GOSUB commands to replace non-standard instructions such as CLS. It must also be borne in mind that some of the machines catered for by BASICODE have very limited memory capacity, and long programs, although they may be perfectly valid in BASICODE, simply will not fit into the available RAM on some machines. It would be perfectly possible to write a program that runs on your own machine but the acid test would lie in SAVEing it and trying to run it on another machine.

Within the main program, the programmer may wish to add certain features that otherwise would not be allowed. This is achieved by adding REM statements that explain precisely what the programmer has in mind. The authors of BASICODE recommend that these statements be enclosed within the lines 20000–24999, although this is not compulsory. After the user has loaded the program, features that are tailored to suit the user's own machine may be added.

Full instructions on the use of BASICODE-2 are provided in the package supplied by the BBC for use in conjunction with the *Chip Shop* series of transmissions. The user receives a cassette with the translation programs for the various machines on side one. Although most computers require loading of a single BASICODE-2 translation program, the BBC Micros and the Vic-20 have separate LOAD and SAVE programs. All of the programs are separated with spoken messages to help the user find the appropriate one. On side two of the cassette are 18 demonstration programs to give some idea of the capabilities of BASICODE.

Given the bewildering variations in what is supposed to be a standard language, the authors of BASICODE have succeeded remarkably well in gathering so many machines under one umbrella.



IAN McKINNELL

The BASICODE-2 manual and cassette is available by sending a cheque or postal order for £3.95 made payable to Broadcasting Support Services to:

Broadcasting Support Services
P.O. Box 7
London
W3 6XJ

RANDOM SELECTION

In the last part of our look at file handling we discussed sequential files. Now we look at an alternative but complementary technique — random access files. Although this type of file offers very direct and therefore faster access to data, it uses more storage space and must be carefully and uniformly defined.

The limitations of sequential files arise because of the necessity to read the information stored in them in order. Random or direct access files provide a solution to these limitations because the records within them can be accessed in any order and very quickly. The word 'random' does not imply that the file is constructed or used in a chaotic manner, it simply means that any segment may be written to or read from without the need to go through all the preceding information.

The obvious problem here is that all files held on cassette tape must be sequential files. There is no way to go straight to an item of data in the middle of a cassette tape; instead the whole tape must be read through. The only way to use random access files on a micro that relies on tape storage is to load all the data into memory, but this limits the file size. Disk drives are needed for useful random access; but even so, a few makes of disk drive cannot support this type of file handling.

The user will also find that random access files are easier to work with than the rather cumbersome techniques needed for sequential files. The division of the file into records and fields that we detailed on page 226 is very important with random access files. To access the file, the required record must be specified. This record, together with its fields, will then be put into a buffer in the computer's memory, where the fields may be deleted, amended or printed.

Fortunately, the operating system will take care of the more complicated structures necessary. It will need to go quickly to the start of a particular record on a disk. It cannot do this on a sequential file, as the only way to locate a record is to read through all the data, counting off each field marker. In order to facilitate rapid location, every record in a random file is the same length. If each record were 100 bytes or characters long, and the program asked for record number 83, the operating system would position the disk head at the start of the 8300th byte of the file. It has a record of how many bytes are in each sector of the disk and can therefore calculate the location of the required record.

This method of file reading may seem complicated, and it is certainly slow, but it is far quicker than reading through a sequential file.

When standardising the length of the files, it is obviously necessary to choose a size that will accommodate the longest record stored in the file. Shorter records must be padded out, usually with spaces (32 in ASCII code). This is a major drawback to random files, as the padding required to make the records up to length is a waste of precious storage space. This means that random files are used for small amounts of information where access needs to be quick, while sequential files are used for bulk storage where access speed is unimportant.

Fields within a record must also be set to a standard length. This is particularly relevant to systems that provide a random access facility to specific fields as well as to particular records. For systems without this facility, it is still a neater and more efficient way of file definition. The first step when designing a random access file is to list the different fields and decide on suitable lengths for them. A field for a person's name should be at least 20 characters long, for example, whereas you would need only two characters in which to store their age.

Economy is crucial when designing a file, as there will inevitably be a trade-off between the amount of information stored and the number of different records. Quite often, coding systems can be devised to reduce the amount of space taken up

Random Vs Sequential Files

	RANDOM ACCESS FILES	SEQUENTIAL FILES
PROS	● Fast access to particular records	● Conserve space ● Available on tape systems
CONS	● Waste space ● Need disks	● Slow and cumbersome
SUITABLE APPLICATIONS	● Predictable data that is in a defined format ● When small numbers of different records are accessed; for example, in a library where customers ask for details of particular books. This is a low 'hit rate' application	● Large quantities of unstructured data ● When most of the records in a file are processed in a single run of the program; for example, in a salary system, where every employee must be paid. This is known as a high 'hit rate'

by data. For example, colour codes 1,2, and 3 for black, red and green, or date codes such as 841011 for 11 October 1984. Coding systems must remain internal to the system, however, and programs should convert the codes back into an easily comprehensible form once a field is displayed or entered.

There are two further considerations to bear in mind when determining record lengths. Most systems place a limit on the maximum length available. This can vary from 128 bytes to as much as 2,048 bytes. Additionally, it is often more efficient to choose a length that is a multiple or factor of the sector size — figures such as 64, 128, 256 or 512 are commonly used. This will prevent individual records from being split over more than one sector and therefore reduces the number of disk calls that need to be made.

Random files are generally much easier to handle than serial files. In both systems, you need to keep an up-to-date count of the number of records in a file and quite often the first record in a random access file (often record number 0) is used to store this information and other relevant information such as the file creation date. The rigid field and record structure would be discarded for this record.

A record can be amended by reading it in, changing it and then writing it back to its location. The record is retrieved by number. Obviously, it is unreasonable to ask a user to remember which record is which by number. So a whole variety of techniques exist for searching and locating particular records, similar in concept to techniques used to search BASIC arrays. Often, one particular field, perhaps a name field, is used as a key to the file. The computer reads in the key field and builds up an index that identifies where various names are stored.

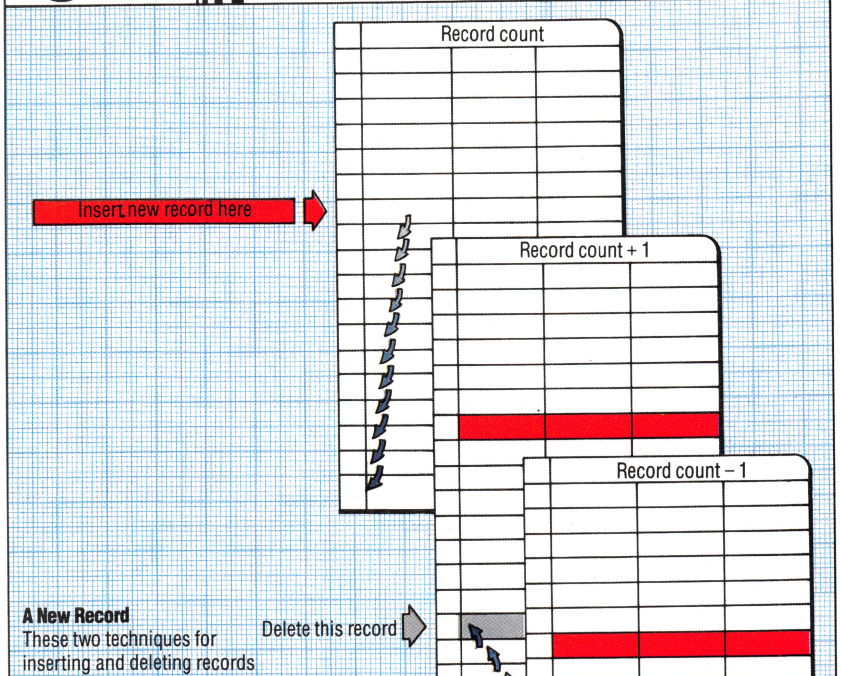
Unindexed random files are often searched record by record just like sequential files. However, if the records are sorted on the key field, fast search methods can be used. Suppose, for example, we wanted to look up 'Jones' in a file sorted by name. We begin by fetching the middle record and discover that the name is 'Phillips'. 'Jones' is before this alphabetically so we can rule out everything after this record. Our next guess is then a record halfway through the first half of the file. The name might be 'Hearst', in which case we need to go forward again and so on. Such techniques can be very sophisticated, and many programs improve performance by keeping large numbers of the most frequently used records in RAM so that they are quickly available. As a result, records can be located and stored within very large files at a high speed.

Deleting and inserting new records can be comparatively slow. The crudest method to delete a record is to copy the record immediately following it into its space, thus overwriting the information in it. Every subsequent record is then copied up one position and finally the record count is reduced by one. In a similar way, a new record can be inserted at any point by moving the last record one number further on and then copying all the records before it and after the number of the new record one space further down. This creates a one-record gap where the new record can be written.

Neither of these techniques is fast, although they are more efficient than similar operations with sequential files. However, insertions and deletions can be made far more quickly if the file has a separate index. When a record is deleted, it is marked as such in the index. The data itself is left unchanged. As new records are added they can be slotted into unused or deleted records and the index updated.

There are two final advantages to consider in the random file system. Firstly, while it is certainly quicker to read and write groups of records together, files can get out of order. Most programs therefore offer a tidy-up facility that sorts records into a logical order and discards deleted records. Secondly, the system of merely marking deleted records as deleted provides a useful safety net, as it is easy to retrieve these records if required. This safety net will operate up to the point when deleted records are overwritten or discarded by a tidy-up program.

Records



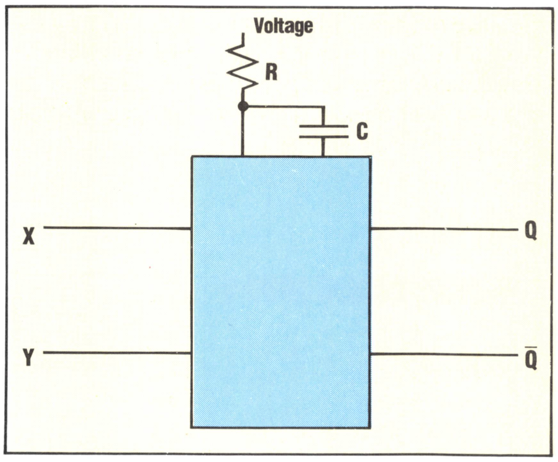
A New Record
These two techniques for inserting and deleting records from a random access file are crude but sufficient for many applications. To insert a record, every record is moved down one record number. To delete, the records are moved up, overwriting the record to be discarded. More advanced techniques use an index to the current and deleted records in a file and therefore avoid any time-consuming copying and moving of records

KEVIN JONES

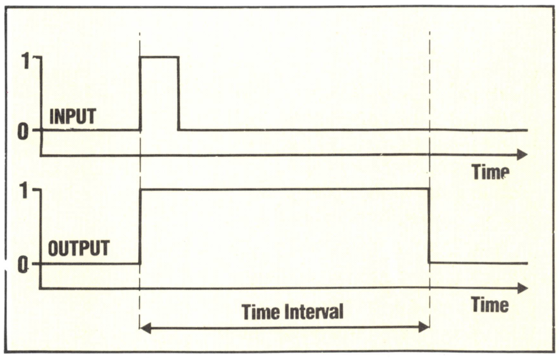
KEEPING TIME

If a computer is to control its many internal functions effectively then accurate timing is required. In this instalment we look at three types of circuit that produce timing signals — monostable circuits, D-type and J-K flip-flops. Later in the course we will investigate the use of these circuits in the design of counters and the CPU.

A *monostable circuit* provides a means of introducing fixed time intervals into logic circuit operations. When a monostable circuit receives a pulse input, the output is set to 1 (HI) for a fixed time interval before returning to its normal zero output (LO) state. The length of time for which the output goes HI is determined by the values of certain components within the circuit. This is an example of a monostable circuit:

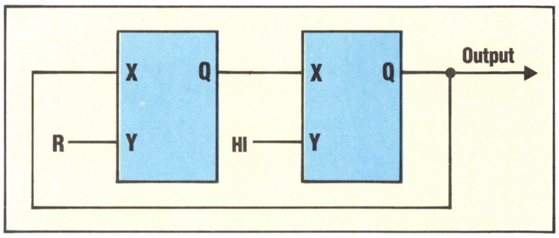


This device may be triggered by changing X from HI to LO or Y from LO to HI. By altering the values of the resistor, R, and the capacitor, C, the output time can be altered. This graph shows how the input and output are related:

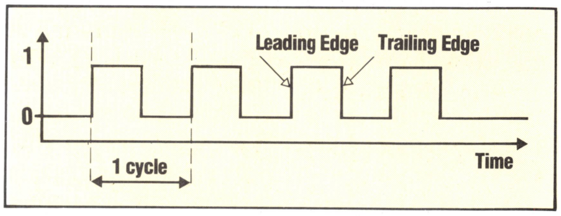


The duration of the HI output could be used to control a tape reader stepper motor or to delay the transmission of a bit for a certain length of time.

Two monostable circuits can be linked together to provide a clock pulse, which oscillates at fixed intervals between HI and LO:



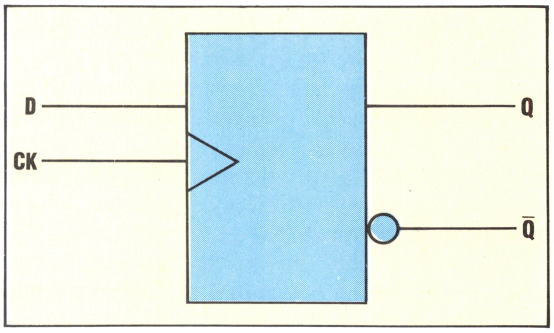
The output produced has a characteristic 'squarewave' appearance (as shown in our graphs). The time interval between the clock output going HI and the next time it goes HI is known as a cycle. Typically this is one millionth of a second. It is this continuous clock signal that is the computer's heartbeat, marshalling the many functions that are carried out in the CPU. The following diagram indicates the names given to the 'edges' of the squarewave graph, where a pulse changes from HI to LO, or vice versa:



Let us now look at two new types of flip-flop whose actions are governed by the regular pulses of the clock.

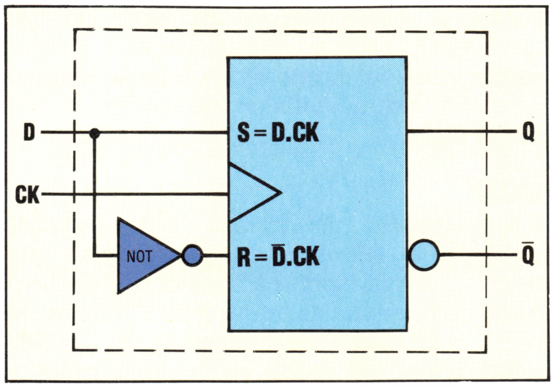
THE D-TYPE FLIP-FLOP

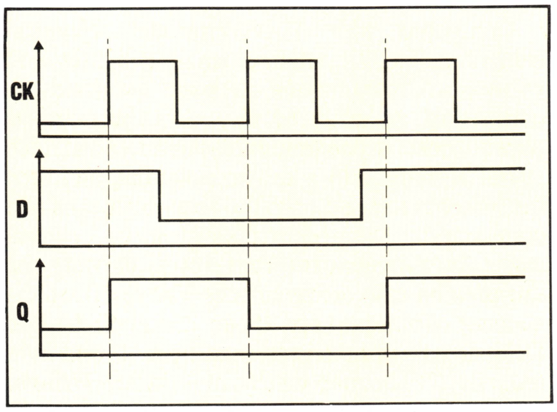
The D-type flip-flop has one logic input (D) and a clock input (CK):



The design of the D-type is based on the R-S flip-flop, which was discussed in the last instalment. It is the addition of the clock input, however, that causes the special method of operation known as *latching*. The output from the circuit, Q, is determined at the start of a clock cycle. If, at this time, the input at D is HI, then the output Q is set

HI. If, however, the input at D is LO, then the output Q is set LO.

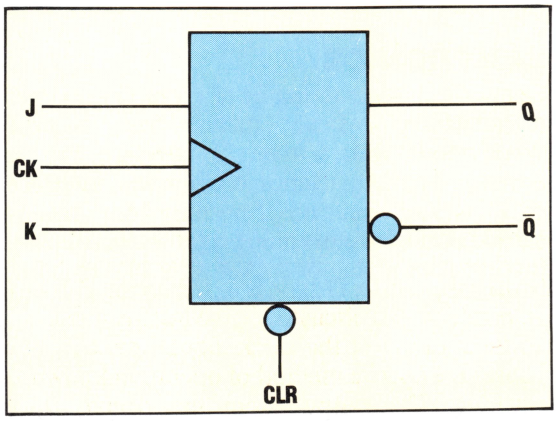




It can be seen from these graphs that the output Q can only change during a LO to HI transition of the clock. Consequently, the D-type is called a 'leading edge triggered' flip-flop.

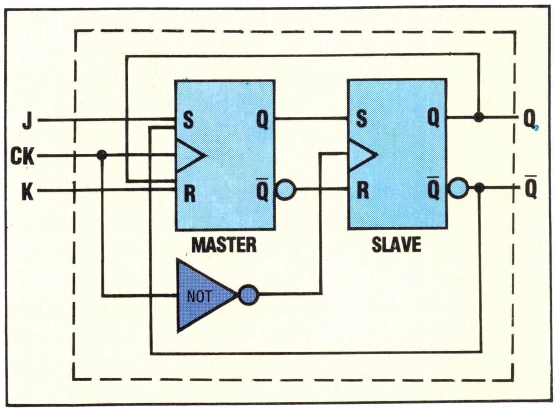
THE J-K FLIP-FLOP

The J-K flip-flop is known as a master-slave device as it comprises, in effect, two R-S flip-flops in a dominant-submissive relationship. A master-slave device allows an input pulse to be stored in one flip-flop, whilst simultaneously giving an output from the other unit dependent on the previous input, all within one clock cycle. An example of this is the shift operation common to most processors, where bits within the register are moved one place to the left or right. Here is the standard circuit diagram for a J-K flip-flop:



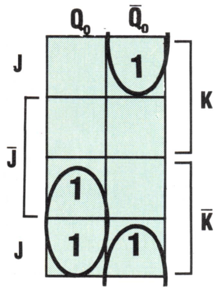
LIZ DIXON

The following diagram shows how the two R-S types are linked together. One is the 'master' and the other the 'slave'. Suppose an input is applied at J or K: if the clock pulse is HI then the input is fed to the master, if the clock input is LO then the slave inputs are fed, since R-S types are leading edge triggered. Thus only one R-S type is activated at any one time, with the previous input being 'locked' inside the other:



In the margin we give a *state table* for the J-K flip-flop. This is similar to a truth table but makes use of a variable, Q_0, the previous output. Notice that HI inputs simultaneously at J and K cause the flip-flop to change state with each clock pulse. This is known as *toggling* and is caused by the feedback of the slave outputs to the master inputs. With an R-S flip-flop this is a disallowed input combination and the output is undefined. By considering Q_0, J and K as inputs, the results from the state table can be placed onto a k-map.

Q_0	J	K	Q
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



From the k-map we can obtain a logic expression:

$Q = \bar{Q}_0.J + Q_0.\bar{K}$

This equation is known as the *characteristic* of the J-K flip-flop.

Answers To Exercise 7 On Page 229

1) A flip-flop is also known as a bistable because it is only stable in one of two states (namely, when Q=1 and $\bar{Q}$=0 or when Q=0 and $\bar{Q}$=1).

2) a) This is not a stable state.

b) The flip-flop will change to the RESET state if the upper input is considered first, or the SET state if the lower gate is considered first.

c) Yes (see answer to b).

d) In order to make all the registers attain a predictable state on power-up they must be either reset or set during the initialising of the computer system.

Erratum
In the diagram of a full adder circuit on page 165 in Issue 9 the annotation was incorrect. The 'sum' and 'carry' labels should be interchanged on each half adder and in the final output. The caption is correct

C

CARRIER TONE

The transmission of computer data via the telephone network requires the use of a modem or acoustic coupler, as a telephone line can handle only a limited range of frequencies. When a link is established between two machines, a constant signal called the *carrier tone* is transmitted. You can hear the carrier tone just before you push the telephone handset into the acoustic coupler, or by picking up the handset during transmission with a modem (the latter is inadvisable, however, since it is liable to introduce noise into the signal and corrupt the data).

Most modems and acoustic couplers have an LED, labelled 'carrier detect', on the front panel. If this should go out then the tone isn't being received and there has been a break in transmission. The frequency of the carrier tone is designed to fall in the middle of a telephone transmission band, and the data is 'modulated' onto this tone according to a predetermined system. Thus a '1' might be represented by a frequency a little higher than the carrier tone, and a '0' by one a little lower. It is this system of modulation that determines the maximum data transmission rates. Most modems can handle up to 1200 baud, but some devices can deal with rates as high as 9600 baud.

In the film *Wargames*, in which a home computer user accidentally breaks into the North American military computer system, the hero made use of an ingenious program for locating computers that can be 'tapped' via the telephone networks. The program that he devised simply dialled up every number for a given area code in sequence. If it answered with a carrier tone the number was logged on disk, otherwise the computer terminated the call and went on to the next number.

CARRY

All arithmetic relies on the idea of digits being 'carried' from one column to another. Computers use exactly the same technique for performing arithmetic on the binary numbers that they process internally. A half-adder is a simple circuit that can add two binary digits together to produce two outputs — a *sum* and a *carry*. A full-adder circuit can cope with a third input as well — a carry-in from the digit on its right. Eight full-adder circuits can therefore be coupled together to add two eight-bit numbers together and produce an eight or nine-bit result. A microprocessor or CPU contains a general purpose arithmetic and logic unit (ALU), and one of its functions is to set a special *carry bit* in one of the internal registers whenever such an addition has resulted in nine binary digits.

CCD

Charge-coupled devices are now replacing conventional television camera tubes to give computers visual input or a 'sense of vision'. This is particularly applicable to the field of robotics, in which researchers are attempting to develop software that will enable an industrial robot to identify components.

Compared with a conventional television camera, a CCD system is smaller, lighter and cheaper to manufacture in large quantities. It consists of an array of tiny electronic circuits packed onto a conventional silicon chip. Each circuit is charged to a known potential, and then the image is focused onto the array by means of a conventional lens. Light falling onto a cell will cause the potential to drain away, allowing the image to be read electronically. The result will be an array of bits, perhaps 256 by 256 at maximum. Dark areas will be represented by a '1' and light areas by '0'. The *threshold level* — the level of light that determines whether a bit will be a 1 or 0 — is determined by the time for which the image is exposed to the array.

It has been discovered that some dynamic RAM chips can behave like CCD arrays if the metal protective covering on the top of the chip is carefully removed. These chips form the basis of the low-cost computer vision systems that are now starting to come onto the market.

CELL

As well as referring to a battery used in a portable computer, *cell* can mean the intersection of a row and column on a spreadsheet. The interesting thing about this type of cell is that it can be used for storing either 'raw' data, such as a figure or the label for a column, or a relationship such as B4+B2, where the contents of the cell are expressed in terms of the values of two other cells.

CENTRONICS

Centronics is the trade name of a computer printer manufacturer whose name has become synonymous with a type of interface that is developed for its machines. A *Centronics* interface is a 'parallel' interface (meaning that data is carried simultaneously along eight separate lines). It consists of a plug or socket with 36 pins on it, although smaller plugs can be used. A piece of equipment with a Centronics interface can be relied on to work with other devices with the same interface. However, because Centronics interfaces send data in one direction only, they are not used on devices such as modems, which need to send and receive data.

PIECE OF THE ACTION

The Apricot is an extremely compact three-piece computer from the British company ACT, which was responsible for the successful Sirius business computer. It comprises the now-familiar processor box, screen and separate keyboard. Although it does not attempt to be a 'true' portable, it is light enough to be moved easily.

Designed very much with the business user in mind, the Apricot comes in two versions: one with two 3½″ disk drives and the other with a single 3½″ drive and a built-in 10 MByte hard disk. Although boasting numerous features designed to appeal to the serious user, the Apricot makes few concessions to the home market — it has no colour graphics, cassette port, games paddles or television output. Supplied as standard, however, is a high-resolution monochrome monitor, a single parallel printer port, a single RS232 serial port, a connector for an optional mouse, some software and a quality keyboard.

The most striking thing about the Apricot is the versatile and innovative keyboard. A novel feature is the Microscreen — a 40-column, two-row LCD display located to the upper right of the main keys. On power-up, the top row of the Microscreen displays the day of the week, the month, the year and the time. The date and time may be altered by using one of the utility programs, and a battery powered clock keeps the time while the computer is not in use.

When the machine is switched on, a test program is automatically started. This displays the amount of memory available (256 Kbytes is standard but this can be upgraded to 768 Kbytes) and asks the user to insert the MS-DOS master disk. For users unfamiliar with operating systems such as CP/M or MS-DOS, a user-friendly 'menu' called the Manager allows easy selection of applications software (such as Supercalc, Multiplan, Microsoft BASIC, etc.) or utilities (such as the keyboard configurator or screen font editor).

The Microscreen is software-controlled, so it acts as more than just a visual display of the time and date. Six user-programmable keys are provided, and the functions allocated to these may be shown at any time on the LCD screen. Thus, when a program displays an option menu on the main screen, the same menu may be duplicated on the Microscreen. Touching the appropriate function key is equivalent to selecting the item from the screen menu by using the Cursor and Return keys. The only criticism here is that ZX81-style membrane keys are harder to use and less positive in action than conventional typewriter-style keys.

Attractive Offer
The ACT Apricot is one of the most attractive looking computers on the market. The machine is also modestly priced for a business micro, yet is built to a high specification. It uses a 16-bit microprocessor with a full 256K of memory as standard, and comes with a high quality monitor

There are also eight ordinary function keys. These are inscribed with legends for their normal functions — HELP, PRINT, MENU, FINISH etc. Like all Apricot keys, however, these can be reconfigured with the supplied Keyedit program. The feel of the Apricot keyboard is up to the high standard expected of a business computer, but the Control and Escape keys are in a slightly odd place. To make the machine easy to move, the keyboard clips to the underside of the main unit. The heavy bulk of the separate monitor is enough, however, to scotch any claim to it being a true portable.

The software supplied with the Apricot is a comprehensive suite of system utilities and Supercalc. 'What if?' - type number crunching is catered for by Supercalc and Superplanner. There is evidence here, as in those other pieces of software adapted for use on the Apricot, of a rather hurried attempt to get it ready in time for the launch of the computer. Two operating systems come with the machine, MS-DOS and CP/M-86. Apricot owners are promised free copies of Concurrent CP/M-86 when it is ready. Only version one of Supercalc is supplied with the Apricot, although versions two and three are available at discount.

One of the earliest criticisms made of the Apricot was that the MS-DOS operating system had been badly implemented and that it was slow in operation. This problem now seems to have been overcome. The applications software supplied with the machine seems to work reasonably quickly, and benchmark programs to

Microfloppy First
Apricot claims to be the first popular business micro to use the new generation of small floppy disks. ACT chose the Sony microfloppy, which uses a floppy disk only 3½" wide, enclosed inside a rigid case. This makes the disk more robust than the traditional 5¼" disks. A spring-loaded cover protects the microfloppy from dust

Apricot XI
The capacity of the microfloppies is limited, so ACT offers the black Apricot XI. This has a 10Mb hard disk built-in that replaces one of the two microfloppies

test Microsoft's MSBASIC are also relatively fast. Even so, the impression given is that the Apricot does not run as quickly as one might hope for a machine with an 8086 processor.

The Apricot's documentation includes an introduction for beginners, a comprehensive guide to the MS-DOS operating system, two useful guides to Supercalc and Superplanner, and extensive manuals for Wordstar and Multiplan. ACT provides little hardware information, although the supplied utilities leave little to be desired for setting up the computer. There are no details on memory mapping or system calls, as might be needed by a software house trying to produce independent software for the Apricot, but this information is readily available from the manufacturers.

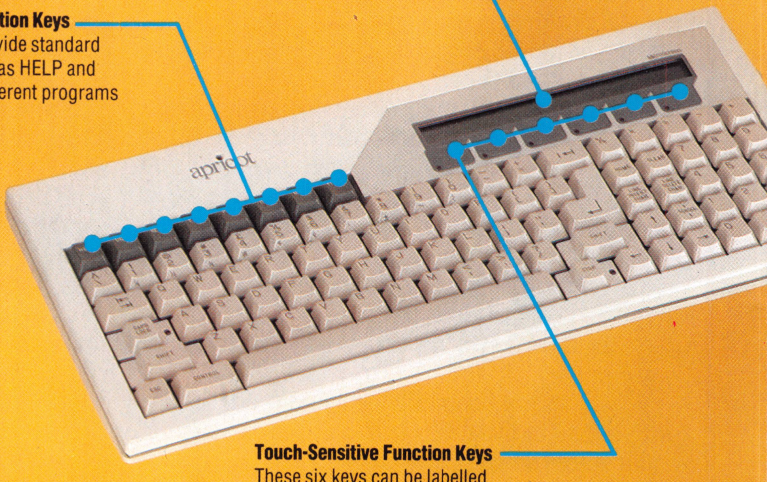
The Apricot has been designed very much with business use in mind — it is not a system for the software engineer or the computer hobbyist. If the Apricot enjoys the success of ACT's Sirius, we can anticipate optional plug-in boards from independent manufacturers, as well as the extra memory boards and modem from ACT itself. Ignoring its merits as a highly versatile and inexpensive business computer, the availability of MS-DOS software on a computer at this price is enough to make the Apricot a very attractive proposition.

Display
A high resolution
phosphor monito
clear and crisp di

LCD Display
A two-line liquid display can be used for messages or as a clock or calculator

Sony Microfloppy
Twin 315K microfloppies are used for compact and convenient storage

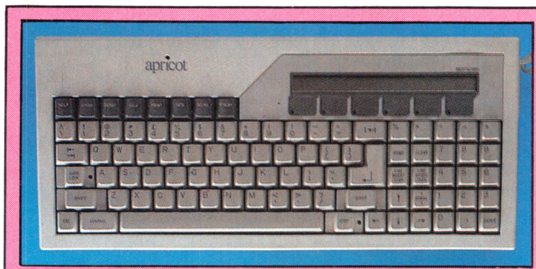
Dedicated Function Keys
These keys provide standard functions such as HELP and REPEAT for different programs



Touch-Sensitive Function Keys
These six keys can be labelled by information displayed on the LCD to match a particular program

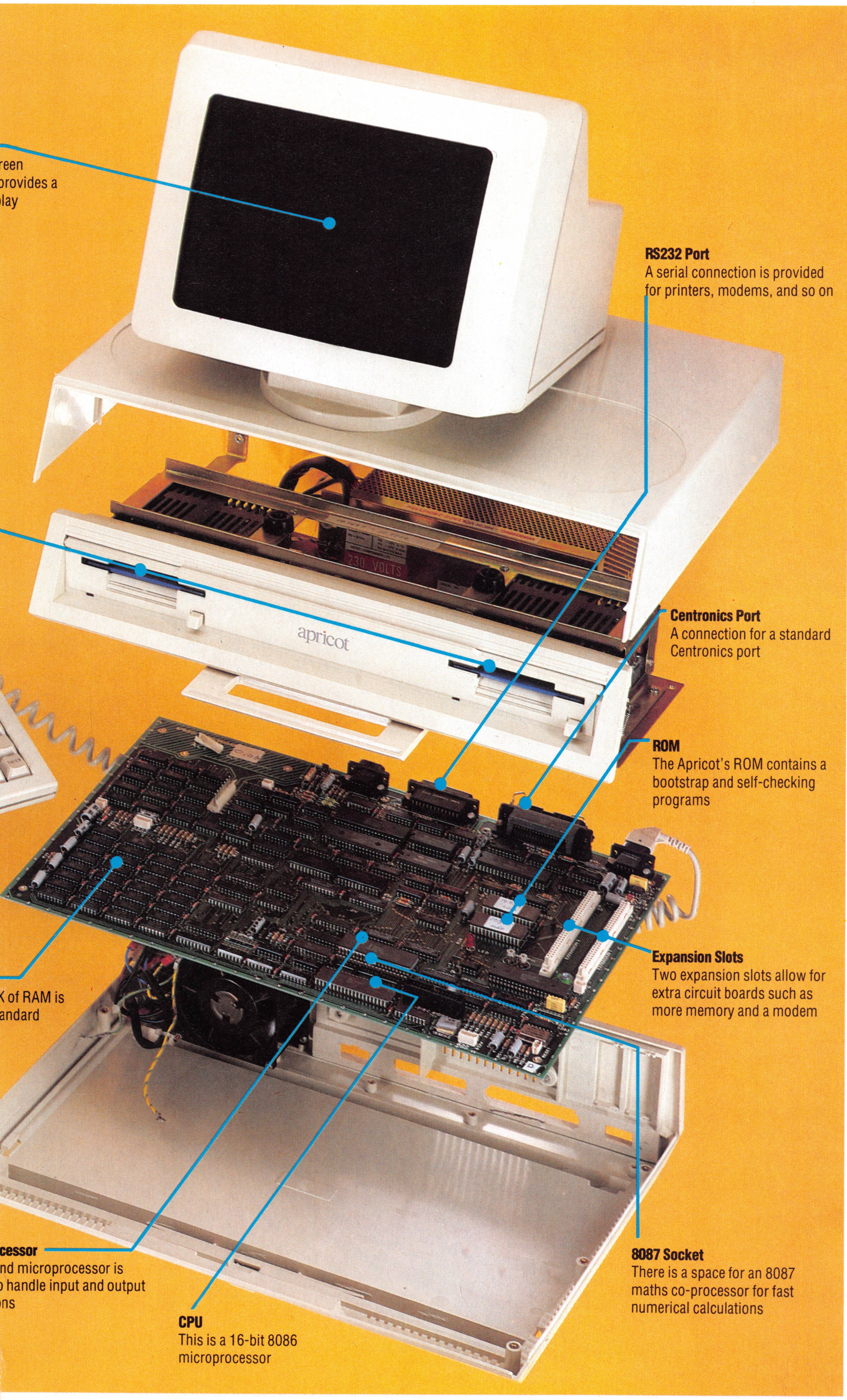
256K RAM
A healthy 25
provided as

I/O P
A se
used
func



Apricot Keyboard
As well as the high quality keys expected on any business micro, the Apricot has six touch-sensitive keys. These are reserved for special functions in various programs. Because these functions change from program to program, the Apricot allows a label to be displayed above each key, thanks to a 40-character by two-line LCD screen. The screen can also be used to display the time

Apricot Monitor
The monitor has a screen only nine inches across, yet has a text display of 132 characters wide by 50 deep, although it is normally used in 80 by 25 character mode. The quality is good and it has a built-in anti-glare screen.
The weight of the monitor reduces the Apricot's claim to be a 'portable', although some users have apparently opted to keep one monitor at home and another at work, simply carrying the main body of the micro to and fro


reen
provides a
lay
RS232 Port
A serial connection is provided for printers, modems, and so on
Centronics Port
A connection for a standard Centronics port
ROM
The Apricot's ROM contains a bootstrap and self-checking programs
Expansion Slots
Two expansion slots allow for extra circuit boards such as more memory and a modem
K of RAM is
andard
cessor
nd microprocessor is
handle input and output
ns
CPU
This is a 16-bit 8086 microprocessor
8087 Socket
There is a space for an 8087 maths co-processor for fast numerical calculations
apricot

CHRIS STEVENS

ACT APRICOT

PRICE
£2,174 (including VAT)
DIMENSIONS
488×413×313 mm
CPU
8086 with option of 8087 maths processor
MEMORY
256K of RAM, expandable to 768K
SCREEN
Text can be shown as 132 × 50 characters or 80 × 25 characters. The graphics resolution is 800 × 400 dots in monochrome only
INTERFACES
Centronics, RS232, 'mouse' socket and internal expansion slots
DISK DRIVES
One or two Sony microfloppies of 315K or 720K capacity each. The Apricot XI model has a 10Mb hard disk and one microfloppy
OPERATING SYSTEMS
MS-DOS, CP/M-86 and Concurrent CP/M-86
KEYBOARD
90 typewriter-style keys plus six touch-sensitive function keys complete with LCD screen for labels
DOCUMENTATION
Comprehensive and well presented
STRENGTHS
A pleasing machine to use that has a better specification than top selling business machines costing noticeably more. It also looks good!
WEAKNESSES
Although good value for money, the Apricot is rather expensive for the hobbyist. It also lacks the ability to use CP/M-80 standard software

ANIMAL MAGIC

Continuing our look at some of the entertaining programs that you can create, we present our own version of the Animals game. This game has always been regarded as fun because it gives the computer an apparent ability to think. However, the principles it employs lie behind many serious artificial intelligence programs.

Animals is a game in which the computer tries to guess the name of the animal that the player is thinking of. It does this by asking questions such as 'Can it fly?', 'Is is furry?' and so on. You are allowed to answer only 'yes' or 'no' and the computer gradually works its way to a point where it can make an 'educated' guess. Obviously, it is quite surprising, especially for people who aren't familiar with computers, that the program is able to do this. The two aspects that make the program particularly entertaining are the computer's ability to communicate in sensible English (even if your own responses are limited to 'yes' and 'no') and the vast store of knowledge that the computer can draw on to guess the animal.

Animals is actually a very simple *heuristic program* — a program that teaches itself to improve its performance as it is running. When the program is first used it 'knows' only two animal names and one question. Depending on whether or not you answer 'yes', it can attempt to guess at what your animal is. If the computer guesses wrong (which it almost certainly will do the first time), the program asks you to enter the name of your animal and a question to distinguish it from the program's guess at your animal. This information is then added to the program's database to build up a 'tree of knowledge' that it can use in the next game. Every time you play the game, the tree increases in size until finally the program is guessing most of your animals correctly and only occasionally discovering a new one.

The interesting point to remember is that the program still doesn't 'know' anything about animals. It is blindly following a guide made up from the combined knowledge of all the players who have played it. The information might as well be about different types of beer, motorcycle parts, medical complaints or your friends and family. A version of the program that allowed you to define the initial question and two answers could be used for a whole variety of different tasks. In other words, it is not the data itself that makes the program work but the way in which it is being organised.

Building the tree with a simple BASIC program is not particularly difficult. Most structures like this are held in BASIC arrays: in this case using T$() for the questions and the names of the animals, and Y() and N() for the links between particular entries in T$. These links form the path through the tree. For any one entry in T$, the corresponding entry in Y() tells the program where to look if the answer to that question is yes. Similarly, the entry in N() is the link for a negative response. At the end of the tree the text in T$() is not a question but the name of an animal. Both Y() and N() are set to 0 in this case and the program has to make a guess that the player is thinking of that particular animal.

This version of the program has been kept short and simple to show you the principles involved. If you want to enhance it, you could improve the presentation for your machine by adding colour graphics, sound and so on. A major improvement would be to give the program some way of storing its database to tape or disk. The best versions of Animals you can find are those that people have been playing for years, and which have built up a vast tree of animals, mythical animals, objects, famous people, friends and so on, all mixed up into one gigantic database. An even more impressive version would enable you to set the first question and alter and edit entries in the tree so that the program would be practical for more serious uses.

Basic Flavours

This program is written in Microsoft BASIC so it should run unchanged on most machines; the only change you might want to make is to the format of the PRINT commands, if you don't like the screen display.

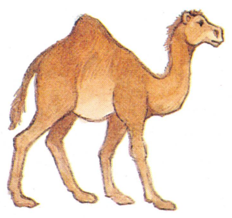
On the Spectrum, all assignment statements must begin with the keyword, 'LET'. Rewrite the following lines as shown:

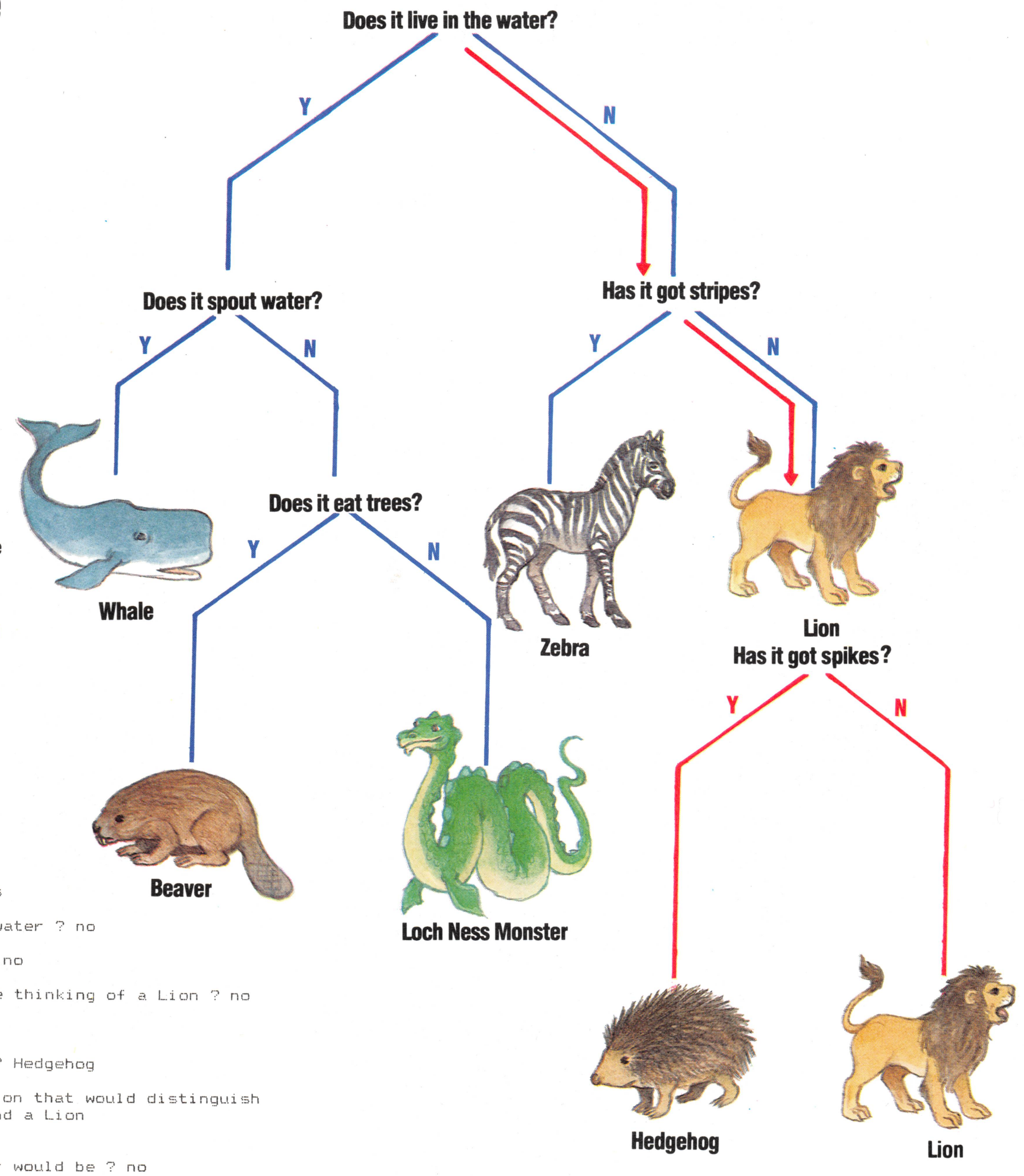
```
45 LET L=40 :REM No. of chars. in a question
50 DIM Y(N):DIM N(N):DIM T$(N,L)
150 LET I$=A$(1):LET P$="A "
200 IF A=30 THEN PRINT:PRINT "BYE":STOP
230 IF Y(P)=0 AND N(P)=0 THEN GOTO 290
```

Learning Tree

This diagram shows the Animals tree after several games have been played. Five different animals have been learnt, with four questions to distinguish between them. From the sample run given (shown in red), you can see how the computer uses the tree to respond to the player's answers in the next game. This time, the player is thinking of a hedgehog and the computer has to learn this new animal when it discovers that the player is not thinking of a lion. The computer then asks for a way to distinguish between a hedgehog and a lion so that it can learn the new animal.

Once you have your own version of the program running, see if you can include a camel





```
Sample Run

Care for a game ? yes

Does it live in the water ? no

Has it got stripes ? no

Is the animal you are thinking of a Lion ? no

I give up!!!

What is your animal ? Hedgehog

Please enter a question that would distinguish
between a Hedgehog and a Lion
? Has it got spikes

For a Lion the answer would be ? no

I now know  6  different Animals !

Care for a game ?
```

```
10 REM  Animals game
20 REM
30 REM ** Set up
40 N=100: REM Max no of animals
50 DIM Y(N),N(N),T$(N)
60 C=3:FOR I=1 TO 3:READ Y(I),N(I),T$(I):NEXT I
70 PRINT :PRINT "A N I M A L S !":PRINT
80 GOTO 190
90 REM ** Answer YES or NO
100 PRINT:PRINT Q$;" ";:INPUT A$
110 IF A$="y" OR A$="Y" OR A$="YES" OR A$="yes" THEN A=1:RETURN
120 IF A$="n" OR A$="N" OR A$="NO" OR A$="no" THEN A=0:RETURN
130 PRINT:PRINT"Please answer YES or NO":GOTO 100
140 REM ** Add A or AN to animal name
150 I$=LEFT$(A$,1):P$=" a "
160 IF I$="A" OR I$="E" OR I$="I" OR  I$="O" OR I$="U" OR I$="a" OR I$
="e" OR I$="i" OR I$="o" OR I$="u" THEN P$=" an "
170 A$=P$+A$:RETURN
180 REM ** Start a new game
190 Q$="Care for a game":GOSUB 100
200 IF A=0 THEN PRINT:PRINT "BYE!":END
210 P=1
220 REM ** Play game
230 IF Y(P)=0 AND N(P)=0 THEN 290
240 Q$=T$(P):GOSUB 100
250 IF A=1 THEN P=Y(P)
260 IF A=0 THEN P=N(P)
270 GOTO 230
280 REM ** Make a guess at the animal
290 A$=T$(P):GOSUB 150:T$=A$
300 Q$="Is the animal you are thinking of"+A$:GOSUB 100
310 IF A=1 THEN PRINT:PRINT"I got it!!!":GOTO 430
320 REM ** Learn a new animal
330 PRINT:PRINT"I give up!!!":PRINT "What is your animal ";:INPUT N$
340 A$=N$:GOSUB 150
350 PRINT:PRINT "Please enter a question that would distinguish" :PRIN
T "between"; A$;" and";T$:INPUT D$
360 Q$="For"+T$+" the answer would be":GOSUB 100
370 A$=T$(P):T$(P)=D$:T$(C+1)=A$:T$(C+2)=N$
380 IF A=1 THEN Y(P)=C+1:N(P)=C+2
390 IF A=0 THEN Y(P)=C+2:N(P)=C+1
400 Y(C+1)=0:N(C+1)=0:Y(C+2)=0:N(C+2)=0
410 C=C+2
420 REM ** End game & loop for another go
430 A=INT(C/2)+1
440 PRINT:PRINT"I now know ";A;" different Animals !"
450 GOTO 190
460 REM ** Initial Data
470 DATA 2,3,"Does it live in the water"
480 DATA 0,0,"Whale"
490 DATA 0,0,"Lion"
```

DAVID HIGHAM

POWERS OF RESOLUTION

Although high resolution graphics do not form part of the Subhunter game being designed during this project, they are an interesting feature of any home computer. There are no easy-to-use high resolution commands in standard CBM BASIC and hence programs written in BASIC that use high resolution routines tend to be very slow.

For low resolution purposes the Commodore 64's screen is divided into 25 rows of 40 character cells, making 1,000 cells in all. As we know, each character is built up from a series of smaller dots, known as pixels, arranged in eight rows; each character cell therefore consists of 64 pixels. For high resolution purposes we need to be able to switch each pixel on or off individually using a single bit in the computer's memory to control each pixel. This idea is known as *bit mapping*. As each memory location contains eight bits, and the screen is made up of 64,000 pixels, 8,000 memory locations are needed to store the high resolution screen information.

The Commodore 64 is switched from standard low resolution mode to high resolution mode by setting bit 5 of location 53265 to one. To set this bit without disturbing any others, the following command should be used:

```
POKE53265,PEEK(53265)OR32
```

Once high resolution mode has been set, then the screen receives its information from an 8,000-byte block of memory. The start of this block of memory is pointed to by location 53272. This is the same location that was used in the construction of user-defined characters in the last article in this series (see page 232).

The area of memory normally assigned to screen memory is used to hold colour information for each eight-by-eight cell of the screen. The 16 colours available on the Commodore 64 can be represented by only four bits; so the upper four bits of any location in the screen memory are used to hold the colour of the pixels that are 'on' within a particular character cell and the lower four bits hold the colour of any pixels turned 'off'. It is therefore possible to have differing pairs of colours, one for foreground and the other for background, in every cell on the screen. If we want a purple background for the whole screen, with high resolution graphics drawn in black, we need the following codes:

Colour code for black is 0 = 0000 in binary
Colour code for purple is 4 = 0100 in binary

Putting the two parts together will give us 00000100, or 4 in decimal. POKEing 4 into every screen memory location (1024 to 2023) will produce the required black graphics on a purple background.

Before we can start to draw on the high resolution screen, the 8,000-byte area that controls what will be seen must be cleared by POKEing a zero to each location. This will take several seconds in BASIC. If this is not done, the screen display will be a mess of dots. This is because that particular memory area takes random values when the machine is switched on.

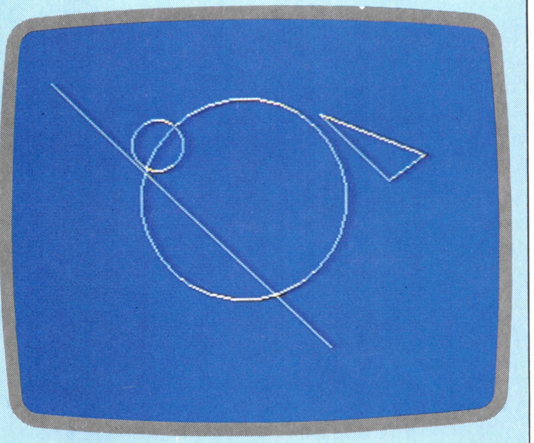
PLOTTING POINTS

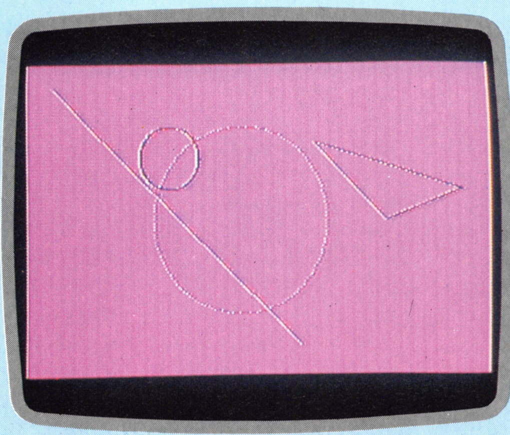
A high resolution graphics program needs to be able to switch on or off individual pixels on the screen. If each point is given an X and a Y coordinate (X and Y are in the ranges 0 to 319 and 0 to 199 respectively) then the program can identify

Relative Speeds

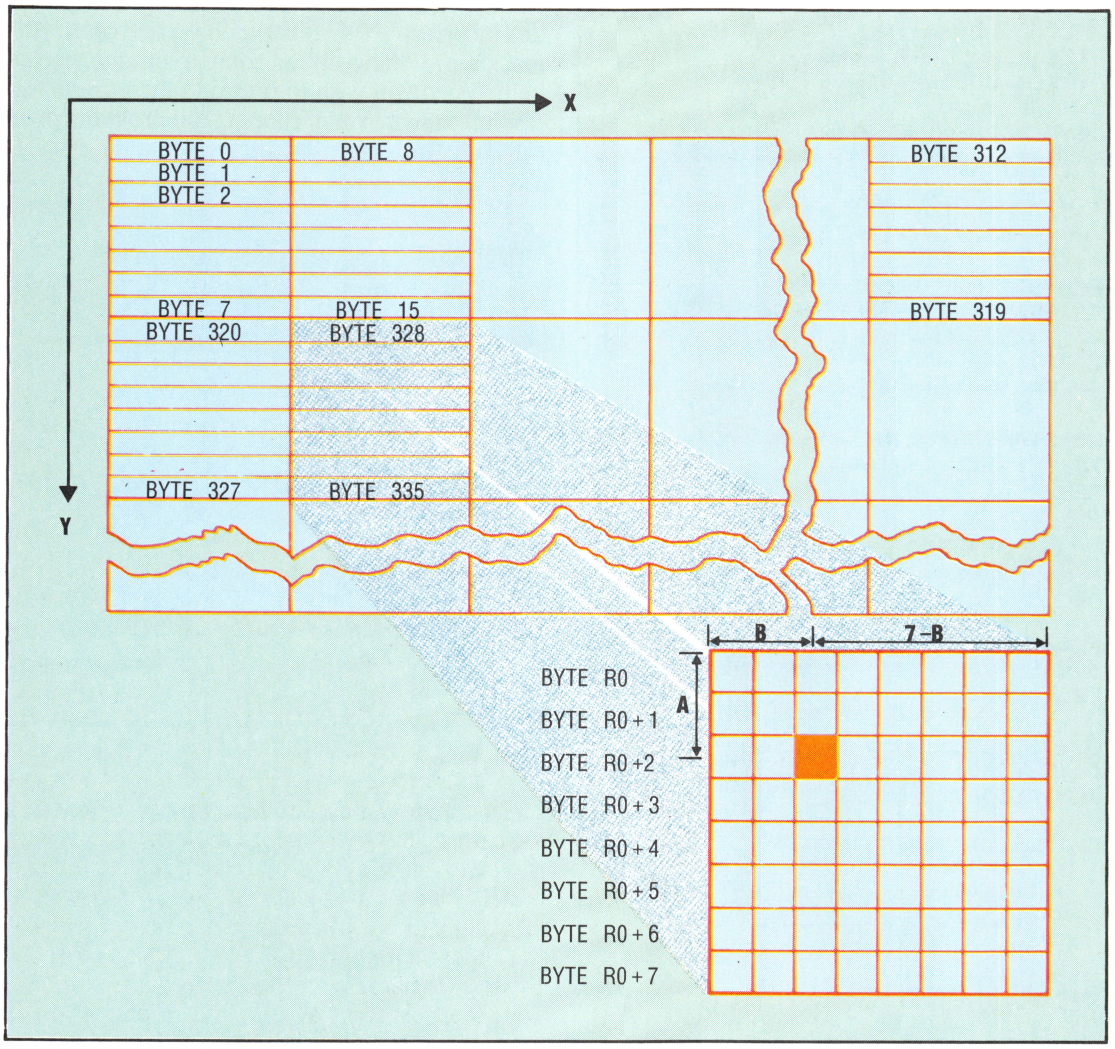
Producing this display took the Commodore 64 nearly 90 seconds and about 50 lines of program. Producing the same display on the Spectrum, however, took about two seconds and the following simple program:

```
4000 REM******HiRes Demo*******
4050 LET N=18: DIM U(N): INK 6: PAPER 1:
BORDER 1: RESTORE 4100
4100 DATA 20,160,160,-160
4120 DATA 80,123,15
4140 DATA 130,90,60
4160 DATA 175,140
4180 DATA 40,-40
4200 DATA 20,15
4220 DATA -60,25
4250 FOR K=1 TO N:READ U(K):NEXT K
4300 PLOT U(1),U(2):DRAW U(3),U(4)
4350 CIRCLE U(5),U(6),U(7)
4400 CIRCLE U(8),U(9),U(10)
4450 PLOT U(11),U(12):DRAW U(13),U(14)
4500 DRAW U(15),U(16):DRAW U(17),U(18)
4600 PAUSE 0
```


EXECUTION TIME = 1.85 SECS


EXECUTION TIME = 89.6 SECS

LIZ HEANEY



Point To Point
The dot pixels that make up the Commodore 64 hi-res screen cannot be accessed directly: the 40 × 25 text screen is mapped onto 8,000 bytes of RAM, each text position being described by eight bytes. A dot pixel's position is described in hi-res by X, its distance (in pixels) from the left of the screen, and Y, its distance from the top of the screen. These numbers must be converted into the address of the byte that contains the pixel, and the number of the relevant bit in that byte

which corresponding bit in the 8,000-byte memory map is to be set to one or zero.

The horizontal byte position can be found from the X co-ordinate by the following command:

```
HB = INT(X/8)
```

Similarly, the required vertical byte can be found from the Y co-ordinate:

```
VB = INT(Y/8)
```

The first byte of the character cell that contains the required bit, R0, can be calculated from HB and VB:

```
R0 = VB*320 + HB*8
```

The byte that contains the required bit will be R0, plus the remainder when Y has been divided by eight. This remainder can easily be found from the right-most three bits of the value of Y. If A = YAND7 and BASE is the address of the first byte in the 8,000-byte block, then the address of the byte, BY, that contains the bit we require can be found:

```
BY = BASE + R0 + A
```

The bit within the byte BY can be found by calculating the remainder when the X co-ordinate is divided by eight. If B = XAND7 then the following POKE will set to one the bit that corresponds to the pixel with co-ordinates X and Y:

```
POKE BY, PEEK(BY)OR(2↑(7–B))
```

Now that each pixel can be individually turned on, routines can be designed to draw shapes on the screen. The following program shows how straight lines can be drawn from one point (X1,Y1) to another (X2,Y2). A circle may be plotted by specifying the co-ordinates of its centre (CX,CY) and the radius RA. There is also a subroutine that will draw a triangle given the co-ordinates of its three corners (XA,YA), (XB,YB) and (XC,YC). You may wish to experiment by entering co-ordinates other than those given in the program.

It is interesting to note that the structure of this program consists of a series of tiered subroutines. The lowest level routine is the one that plots a single point on the screen. This subroutine is used by a higher level routine that draws a straight line. At a higher level still, the PLOT TRIANGLE routine uses the PLOT LINE routine three times to construct its three sides. This approach to programming has many advantages. It is flexible, since it would be very easy to design a routine to draw, say, regular hexagons. Such a routine would call the PLOT LINE routine, which would in turn call the PLOT POINT routine. Alternatively, the DRAW TRIANGLE routine could be used to construct hexagons from equilateral triangles. In this case the DRAW HEXAGON routine would form a fourth tier to the program structure.

```
65 REM  **** HI-RES DEMO ****
70 PRINT CHR$(147): REM CLEAR SCREEN
80 POKE 53280,0 : REM COLOUR BORDER BLACK
90 :
100 REM **** COLOUR SCREEN MEMORY AREA ****
110 FOR I=1024 TO 2023: POKE I,4: NEXT I
120 :
130 REM **** SET BIT MAP POINTER ****
140 BASE =8192: POKE 53272. PEEK(53272) OR 8
150 :
160 REM **** CLEAR BIT MAP MODE ****
170 FOR I=BASE TO BASE + 7999:POKE I,0:NEXT I
180 :
190 REM **** SET BIT MAP MODE ****
200 POKE 53265, PEEK(53265) OR 32
210 :
220 REM **** DRAW STRAIGHT LINE ****
230 X1=20: X2=190: Y1=15: Y2=180
240 GOSUB 800: REM PLOT LINE
250 :
300 REM **** DRAW CIRCLE ****
310 CX=150: CY=100: RA=60
320 GOSUB 900: REM PLOT CIRCLE
330 :
370 **** ANOTHER CIRCLE ****
380 CX=100: CY=60: RA=20
390 GOSUB 900: PLOT CIRCLE
400 :
410 REM **** DRAW TRIANGLE ****
420 XA=200:XB=250:XC=300:YA=50:YB=100:YC=80
430 GOSUB 600: REM PLOT TRIANGLE
440 :
450 GOTO 450: REM END OF MAIN PROGRAM
460 :
470 :
600 REM **** PLOT TRIANGLE SUBROUTINE ****
610 :
620 X1=XA: X2=XB: Y1=YA: Y2=YB
630 GOSUB 800: REM PLOT LINE
640 X1=XB: X2=XC: Y1=YB: Y2=YC
650 GOSUB 800: REM PLOT LINE
660 X1=XC: X2=XA: Y1=YC: Y2=YA
670 GOSUB 800: REM PLOT LINE
680 RETURN
690 :
800 REM ***** PLOT LINE SUBROUTINE *****
810 S=1
820 IF X2<X1 THEN S=-1
830 FOR X=X1 TO X2 STEP S
840 Y=(Y2-Y1)*(X-X1)/(X2-X1)+Y1
850 GOSUB 1000: REM PLOT POINT
860 NEXT X
870 RETURN
880 :
900 REM **** PLOT CIRCLE SUBROUTINE ****
910 :
920 FOR ANGLE = 0 TO 2*PI STEP .04
930 X=INT (RA*COS(ANGLE)+CX)
940 Y=INT (CY-RA*SIN(ANGLE))
950 GOSUB 1000: REM PLOT POINT
960 NEXT ANGLE
970 RETURN
980 :
1000 REM **** PLOT POINT SUBROUTINE ****
1010 :
1020 IF X>319 OR X<0 OR Y>199 OR Y<0 THEN
GOTO 1070
1030 HB=INT(X/8): VB=INT(Y/8)
1040 RO=VB*320+HB*8: A= Y AND 7: B=X AND 7
1050 BY=BASE+RO+A
1060 POKE BY, PEEK(BY)OR(2^(7-B))
1070 RETURN
```

Be sure to SAVE this program before running it, as an incorrect POKE instruction can cause the machine to 'hang-up' or come to an unexpected halt. When you wish to return to the normal low resolution screen after running this program, press the Run/Stop and Restore keys simultaneously.

Subhunter Program

An important part of the subhunter game we are designing is the routine that updates the score. There are many ways of allocating scores during a game of this type; our scoring system will be based on the following rules.

1) The depth and speed of the sub are important factors. A fast-moving sub at depth is more difficult to hit than a slow sub near the surface. The score allocated to any sub will take account of this.

2) If the sub is hit, its value is added to the player's score, but if the sub reaches the edge of the screen unharmed, its value is subtracted from the player's score. No negative scores will be allowed.

Later in the project, we will deal with the routine that randomly selects the speed and depth of a sub, but all we need to know for now is that the sub's depth is stored in the variable Y3, and its speed is stored in DX. The sub's value can be calculated on this basis. To ensure that only whole number sub values are calculated, the INT function is used as follows:

Sub value = INT(Y3+DX*30)

We will store the player's current score in a variable SC. All that remains to be done is either to add or subtract the sub value from SC according to whether the sub was hit or it escaped. The UPDATE SCORE subroutine is used by two program sections:

1) Where the sub's position is tested to see if it has reached the edge of the screen and
2) during the HIT routine.

The flag DS can be set during these two parts of the program to indicate which part is using the UPDATE SCORE subroutine. Setting DS = 1 in the HIT routine and DS = −1 in the EDGE OF SCREEN routine, the score can be increased or decreased by the sub value as follows:

SC = SC + INT(Y3+DX*30)*DS

After making sure that the score has not dropped below zero, the new value of SC can be PRINTed to the top line of the screen. Add these lines to your program and SAVE it.

```
5500 REM **** UPDATE SCORE ****
5510 SC=SC+INT(Y3+DX*30)*DS
5520 IFSC<0THEN SC=0
5530 PRINT CHR$(19);CHR$(144);" SCORE ";
     SC;CHR$(157);" "
5540 RETURN
```

In the next section we will look at the creation of the sprites that will be used for the ship, sub, depth charges, and explosion.

LAST IN FIRST OUT

The stack is a defined area of computer memory attached to the CPU that acts as a convenient workspace and takes a vital part in subroutine execution. It is easily accessed through the stack instructions, which permit the quick copying and restoring of register contents. We examine the stack and its operation in detail here.

Memory management is the essence of Assembly language programming, and most of the instructions we've studied so far in the course are concerned with simply loading data to or from memory locations. These locations have been accessed in a variety of ways — the addressing modes — but the instructions concerned have always taken a specific memory address as part of the operand. There is a class of instructions, however, that access a specific area of memory but do not take an address as operand. These instructions operate on the area of memory known as the *stack,* and they are known as the stack operations.

The stack is provided for both the central processing unit and the programmer to use as temporary workspace memory. It is a kind of 'scratch-pad', easily written to, read from and erased. The stack operations copy data from the CPU's registers into vacant areas of the stack, or copy data from the stack back into the CPU registers. These instructions require no address operand because a specified CPU register, the *stack pointer,* always contains the address of the next free stack location. Thus, anything written to the stack is automatically written to the byte pointed to by the stack pointer, and data loaded from the stack is always copied from the stack location last written to. Whenever a stack operation is executed, the stack pointer is adjusted as part of the operation.

In 6502 systems the stack is the 256 bytes of RAM from $0100 to $01FF; in Z80 systems the location and size of the stack are determined by the operating system, but may be changed by the programmer. This variation reflects the differences in the internal organisation of the two microprocessors (see the diagram on page 136): the 6502 has a single-byte stack pointer, while the Z80 stack pointer consists of two bytes.

The contents of the 6502 stack pointer are treated by the CPU as the lo-byte of the stack address, and a hi-byte of $01 is automatically added to this by means of a 'ninth bit' wired into the stack pointer. This extra bit is always set to one, so 6502 stack addresses are all on page one.

The Z80 stack pointer is a two-byte register capable of addressing any location between $0000 and $FFFF — the entire address space of the Z80 itself. The stack can thus be located anywhere in RAM, and its location can be changed by the programmer. This is not recommended, however, since the operating system initially sets the stack location and stores data on it. As the operating system may interrupt the execution of any machine code program at any time, and expect to find data relevant to its operation on the stack, any alteration of the location of the stack will mean that the data will not be available to it and the system may crash.

As an example of the use of the stack, consider the following routine to exchange the contents of two memory locations, LOC1 and LOC2:

6502		Z80	
LDA	LOC1	LD	A,(LOC1)
PHA		PUSH	AF
LDA	LOC2	LD	A,(LOC2)
STA	LOC1	LD	(LOC1),A
PLA		POP	AF
STA	LOC2	LD	(LOC2), A

The contents of LOC1 are loaded into the accumulator, and from there copied or 'pushed' onto the stack. The contents of LOC2 are then loaded to the accumulator, and stored in LOC1. The contents of the top byte of the stack are then copied or 'popped' into the accumulator, which restores the original contents of LOC1 to the accumulator. This is copied to LOC2, and the exchange is complete. Notice that the stack operations 'saved' the contents of LOC1 in memory as long as needed, but without the program specifying any memory location — except, by implication, the next free location on the stack.

This program fragment shows us a lot about stack operations. Primarily, they are reciprocal and sequential. The last item pushed onto the stack is retrieved by the next pop from the stack. Successive pushes with no intervening pops write data into successive stack locations, one 'above' the other, while pops without intervening pushes access successive locations 'downwards' from the current 'top' of the stack.

To visualise the stack, imagine writing notes on postcards and stacking them next to you on the desk, then reading and discarding cards until the stack is empty. The most recently written of the cards remaining in the stack is always the one on top. For this reason the stack is known as a Last In First Out (LIFO) data structure. Its converse, a First In First Out (FIFO) structure, is a queue. It is conventional to talk about the next free byte in the stack as the *top* of the stack, and to imagine the stack growing upwards. In both the Z80 and the 6502, however, the stack pointer is decremented by a push, so that the stack top is actually at the lower memory address than the stack bottom. This is less confusing if we describe the stack as 'rising towards zero'.

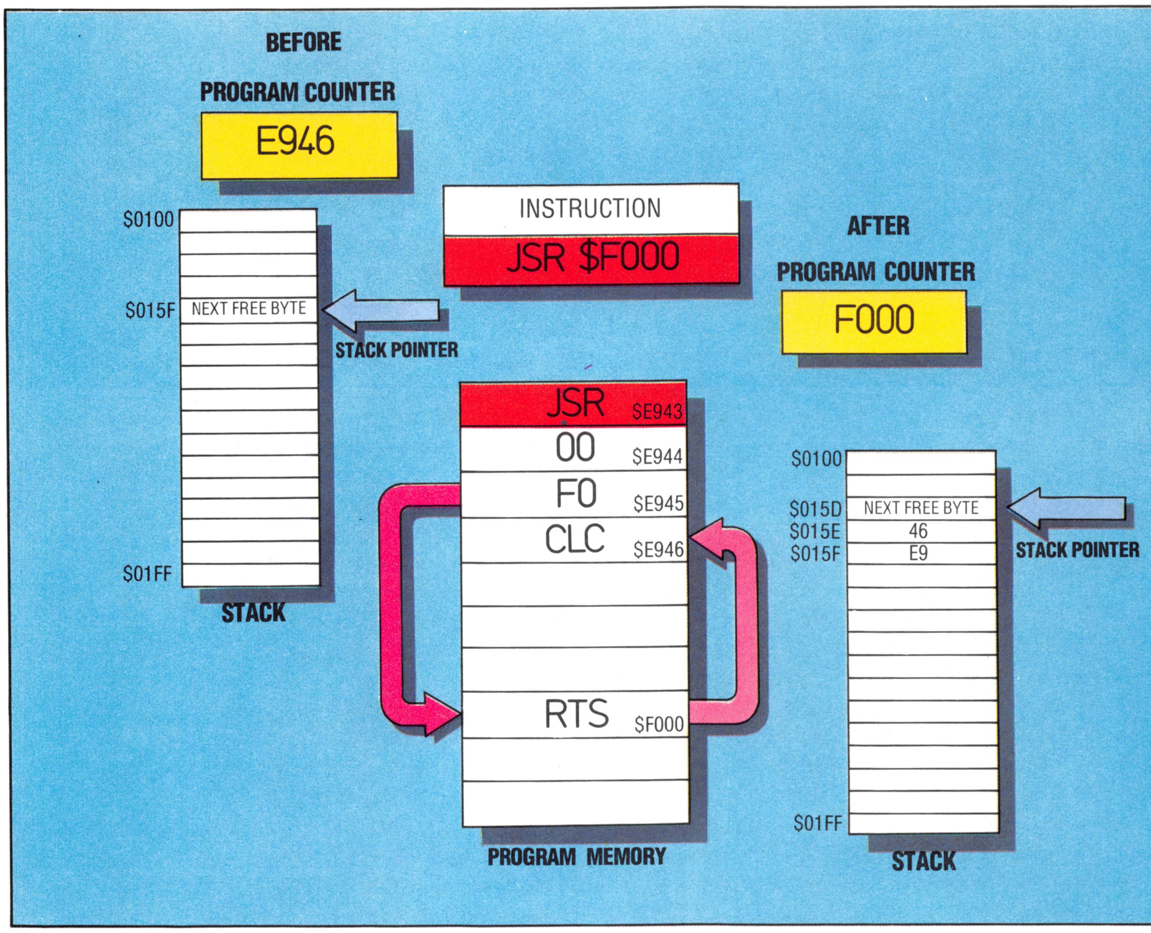
The first program fragment is also typical of programs using the stack in that the number of push instructions is exactly counterbalanced by the number of pops. This is not essential, but failure to observe this harmony of opposites when writing subroutines may result in an incorrect return from the subroutine and consequent program failure. This is one of the commonest bugs in Assembly language programs, but can be fairly easily traced by comparing the number of pop and push instructions in a program.

The Z80 version of the routine differs noticeably from the 6502 in one major respect: the 6502 pushes only single-byte registers onto the stack, while the Z80 always pushes a two-byte register. When you push or pop the Z80 accumulator, you also push or pop the contents of the processor status register, because the CPU treats these two single-byte registers as one two-byte register called the AF (accumulator-flag) register. The power of the Z80 derives greatly from its ability to handle two-byte registers.

It is a good programming habit to start subroutines by pushing the contents of all CPU registers onto the stack, and popping them off the stack immediately before returning from the subroutine. This ensures that the CPU after the subroutine call is in exactly the same state as it was before it, and means that any of the registers can be used in the subroutine with no fear of corrupting data essential to the main program. For example, consider this program subroutine:

	6502		Z80	
	LDA	LOC1	LD	A,LOC1
SUM	ADC	#$6C	ADC	A,$6C
GSUB	JSR	SUBR0	CALL	SUBR0
TEST	BNE	SUM	JR	NZ,SUM
EXIT	RTS		RET	
SUBR0	PHP		PUSH	AF
	PHA		PUSH	HL
	TXA		PUSH	DE
	PHA		PUSH	BC
	TYA		PUSH	IX
SUBR1	PHA		PUSH	IY
SUBR2	STA	LOC2	LD	(LOC2),A
	LDA	#$00	LD	A,$00
SUBR3	PLA		POP	IY
	TAY		POP	IX
	PLA		POP	DE
	TAX		POP	BC
	PLA		POP	HL
SUBR4	PLP		POP	AF
	RTS		RET	

Here, the effect of the instructions between SUBR0 and SUBR1 is to push the current register



The Stack

The Stack is essential in handling subroutine calls (such as the 6502's JSR). When the 'jump to subroutine' instruction is first loaded, the program counter contains the address of the next instruction in the program ($E946 here, the address of the CLC instruction); this address is 'pushed' onto the stack causing the stack pointer to be changed to point to the next free byte of stack space ($015D here), and the operand of the jump instruction ($F000 here) is loaded into the program counter as the address of the next instruction, thus causing a branch in program flow

KEVIN JONES

contents onto the stack, and the effect of the instructions between SUBR3 and SUBR4 is to restore those contents to the registers. The substantive instructions in the subroutine are the two starting at SUBR2, but the second of these is ineffective since the subsequent instructions completely change the state of the accumulator.

Notice that the Z80 PUSH and POP instructions can take any of the register pairs as an operand, but the 6502 can operate on only the accumulator (PHA and PLA) and the processor status register (PHP and PLP). Hence the need for the register-accumulator transfers (TXA, TAX, TYA, TAY) in the 6502 version. Notice also that we have made a deliberate mistake in the Z80 version in not 'popping' all of the registers in the reverse order to which they were 'pushed'. It illustrates the care needed in stack operations, but also demonstrates that you can push the stack from one register and then pop that value off the stack back into a different register — a laborious but sometimes convenient way of doing data transfers between registers.

The functions and uses of the CPU registers are the subjects of the next instalment, in which we conclude our general examination of the Assembly language instruction set. We also begin the study of machine code arithmetic.

Exercises

1) Rewrite the second routine given in the answers to the previous exercises so that the message at LABL1 is stored back at LABL1, but in reverse order, thus:

```
LABL1 EGASSEM A SI SIHT
```

Use the stack for this reversal.

2) Develop this routine so that the words of the message remain in their original order, but the characters of each word are reversed, thus:

```
LABL1 SIHT SI A EGASSEM
```

Answers To Exercises On Page 237

1) This subroutine stores the numbers $0F to $00 in descending order in the block of $10 bytes reserved by the DS pseudo-op at LABL1.

6502		Z80	
ORIGIN	ORG $7000	ORIGIN	ORG $C000
LABL1	DS $10	LABL1	DS $10
LABL2	DW $7100	LABL2	DW $C100
		OFFST	EQU $0F
BEGIN	LDY #$FF	BEGIN	LD IX,LABL1
	LDX #$10		LD B,OFFST
LOOP0	INY	LOOP0	LD (IX+0),B
	DEX		INC IX
	TXA	ENDLP0	DJNZ LOOP0
	STA LABL1,Y		LD (IX+0),B
ENDLP0	BNE LOOP0		RET
	RTS		

The differences in approach and instructions between the Z80 and 6502 are revealing. The 6502 uses the Y register as an index to the address LABL1, and the X register as a loop counter and source of the data to be stored. Notice that the X register is decremented two instructions before the BNE test at ENDLP0, but because TXA (Transfer X contents to the Accumulator) and the STA do not affect the processor status register, the test works on the effects of decrementing X.

The Z80 version uses IX indirect addressing mode to hold the storage address, and uses the B register as the counter and source of data. At ENDLP0 in the Z80 version we see DJNZ LOOP0, meaning 'decrement register B, and jump relative to LOOP0 if the result is non-zero'. This instruction is almost an Assembly language FOR...NEXT structure, and certainly makes writing Z80 loops easy and convenient.

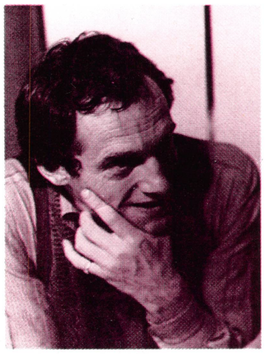
2) This routine copies the message stored at LABL1 to the block starting at the address stored at LABL2. The value $0D (the ASCII code for Return or Enter) is stored at the end of the message as a terminator.

6502		Z80	
ORIGIN	ORG $7000	ORIGIN	ORG $C000
LABL1	DB 'THIS IS A MESSAGE'	LABL1	DB 'THIS IS A MESSAGE'
TERMN8	DB $0D	TERMN8	DB $0D
LABL2	DW $7100	LABL2	DW $C100
CR	EQU $0D	CR	EQU $0D
ZPLO	EQU $FB		
BEGIN	LDA LABL2	BEGIN	LD IX,LABL1
	STA ZPLO		LD IY,(LABL2)
	LDA LABL2+1	LOOP0	LD A,(IX+0)
	STA ZPLO+1		LD (IY+0),A
	LDY $FF		INC IX
LOOP0	INY		INC IY
	LDA LABL1,Y		CP CR
	STA (ZPLO),Y	ENDLP0	JR NZ,LOOP0
	CMP CR		RET
ENDLP0	BNE LOOP0		
	RTS		

The 6502 version uses the Y register as an index to the indirect address ZPLO, in the post-indexed indirect mode. This mode is possible only with the Y register, and requires a zero page operand address — hence the initialisation of ZPLO and ZPLO+1 with the address stored at LABL2. The operating system in 6502 machines uses most of the zero page locations, but locations $FB to $FF on the Commodore 64, and $70 to $8F on the BBC Micro, are unused, so ZPLO is set to one of these locations. The Z80 version uses IX in indexed mode, and IY in indexed indirect mode.

Both routines use a 'compare the accumulator' instruction — CMP CR (6502) and CP CR (Z80) — in which the operand is subtracted from the accumulator contents, thus affecting the processor status register (PSR) flags. The accumulator contents are then restored, while the PSR shows the results of the comparison. When the accumulator contains $0D (the message terminator), the result of the comparison will be that the zero flag is set. Thus the ENDLP0 test will fail and control will pass to the return instruction.

THE CAMBRIDGE CONNECTION


John Shirreff

The city of Cambridge is fast becoming an English 'Silicon Valley'. Not only are successful companies such as Acorn and Sinclair Research based there, but also lesser-known firms like Camputers, which is now securing remarkable sales of its Lynx microcomputer in countries as diverse as Greece, Norway, Jordan and South Africa.

Camputers began as the brainchild of one man. In 1976, Dick Greenwood started working as a freelance electronics designer, accepting contracts from firms such as Pye Telecoms. By the end of the 1970s, Greenwood had formed his own company and had become involved in more specialised development work, also on a contractual basis. The company diversified into software development, producing a Bar Management System — a stock control package that enabled breweries to monitor their sales in pubs and off-licences.

The company then moved into microcomputer design, concentrating on projects based around the Zilog Z80 chip. In February 1981, Greenwood set up Camtronic Circuits (later to become Camputers), and with the help of the government's Small Firms Loan Guarantee Scheme work began on the Lynx home computer in the summer of 1981. Greenwood's stated aim was to 'teach the Z80A to dance around problems, not barge through them'.

In charge of the hardware development side of the project was John Shirreff, a graduate of Cambridge University, who joined the company after working in the rock music industry. Software development was handled by Davis Jansons, who wrote the version of BASIC used by the machine.

When it appeared in 1982, the Lynx was considered a very professional-looking machine. Packaged in an attractive grey casing, with a full QWERTY-style typewriter keyboard, the computer came equipped with a standard 48 Kbyte memory that could be expanded to 192 Kbytes. The machine had the ability to display up to eight different colours, with a high resolution mode of 248 × 256 pixels. It also had a built-in speaker to take full advantage of its audio capabilities.

Unfortunately, the Lynx never really became popular in the United Kingdom, although sales abroad encouraged the company to continue to develop the basic design. The Lynx 96 appeared shortly afterwards, and came equipped with over 37 Kbytes of user RAM, as well as the ability to run $5^1/_4''$ floppy disk drives. Furthermore, the Lynx 96 came with pre-set sound effects on board. Serial and parallel printer interfaces were available for the machine as optional extras.

More recently, the company has introduced a machine aimed at the small business side of the microcomputer market. The computer is called the Lynx Laureate, and as it is based around the Z80 microprocessor, it can run under the CP/M operating system, which gives the user access to the vast quantity of CP/M software that has been written in the last decade. This is an important selling point as few users nowadays would consider buying a computer, especially for business use, that lacks adequate software support. Although designed as a small business machine, the Laureate is nevertheless compatible with the other Lynx models and is fitted with a 40-way expansion bus that allows it to use the full range of Lynx peripheral packs, including a parallel printer, joystick and ROM-based software.

Despite this impressive range of available micros, Camputers, under its present chairman Stanley Charles, is continuing to develop new products. In the near future the company is planning to launch an alternative version of the Laureate business machine. This system will be available as an integrated package, as opposed to the modular layout of the present design. There are also plans for a UK relaunch of the Lynx 48, under the name Leisure. This will be aimed specifically at the home and games computer market — an area in which the company feels its machines have been unfairly neglected. Looking further ahead, Camputers is working on a machine that the company expects will be fully competitive with the Sinclair QL.

Business System
Shown here is the Lynx Laureate modular system designed for business users. 128K of memory on-board enables it to support CP/M

COURTESY OF CAMPUTERS

Mentathlete

Home computers. Do they send your brain to sleep – or keep your mind on its toes?

At Sinclair, we're in no doubt. To us, a home computer is a mental gym, as important an aid to mental fitness as a set of weights to a body-builder.

Provided, of course, it offers a whole battery of genuine mental challenges.

The Spectrum does just that.

Its education programs turn boring chores into absorbing contests – not learning to spell 'acquiescent', but rescuing a princess from a sorcerer in colour, sound, and movement!

The arcade games would test an all-night arcade freak – they're very fast, very complex, very stimulating.

And the mind-stretchers are truly fiendish. Adventure games that very few people in the world have cracked. Chess to grand master standards. Flight simulation with a cockpit full of instruments operating independently. Genuine 3D computer design.

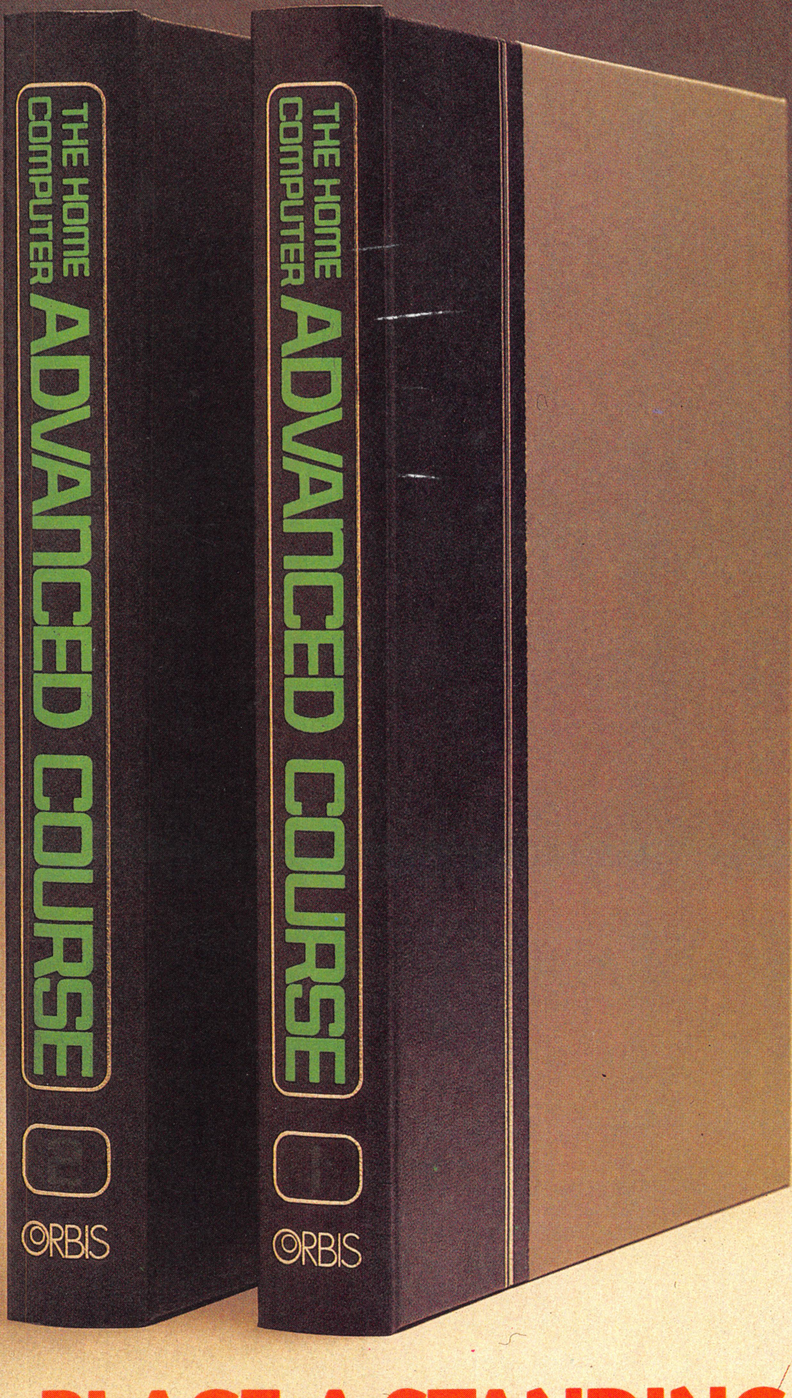
No other home computer in the world can match the Spectrum challenge – because no other computer has so much software of such outstanding quality to run.

For the Mentathletes of today and tomorrow, the Sinclair Spectrum is gym, apparatus and training schedule, in one neat package. And you can buy one for under £100.

sinclair

THE HOME COMPUTER ADVANCED COURSE

WE HAVE DESIGNED BINDERS SPECIALLY TO KEEP YOUR COPIES OF THE 'ADVANCED COURSE' IN GOOD ORDER.


THE HOME COMPUTER ADVANCED COURSE
ORBIS
THE HOME COMPUTER ADVANCED COURSE
ORBIS

All you have to do is complete the reply-paid order form opposite – tick the box and post the card today – **no stamp necessary!**

By choosing a standing order, you will be sent the first volume free along with the second binder for £3.95. The invoice for this amount will be with the binder. We will then send you your binders every twelve weeks – as you need them.

Important: This offer is open only whilst stocks last and only one free binder may be sent to each purchaser who places a Standing Order. Please allow 28 days for delivery.
Overseas readers: This free binder offer applies to readers in the UK, Eire and Australia only. Readers in Australia should complete the special loose insert in issue 1 and see additional binder information on the inside front cover. Readers in New Zealand and South Africa and some other countries can obtain binders now. For details please see inside the front cover. Binders may be subject to import duty and/or local tax.
The Orbis Guarantee: If you are not entirely satisfied you may return the binder(s) to us within 14 days and cancel your Standing Order. You are then under no obligation to pay and no further binders will be sent except upon request.

PLACE A STANDING ORDER TODAY.

MORAL